The Performance of PC Solid-State Disks (SSDs) as a Function of Bandwidth, Concurrency, Device Architecture, and System Organization

Cagdas Dirik* and Bruce Jacob
Dept. of Electrical and Computer Engineering
University of Maryland, College Park
{cdirik, blj}@umd.edu

ABSTRACT

As their prices decline, their storage capacities increase, and their endurance improves, NAND Flash Solid State Disks (SSD) provide an increasingly attractive alternative to Hard Disk Drives (HDD) for portable computing systems and PCs. This paper presents a study of NAND Flash SSD architectures and their management techniques, quantifying SSD performance under user-driven/PC applications in a multi-tasked environment; user activity represents typical PC workloads and includes browsing files and folders, emailing, text editing and document creation, surfing the web, listening to music and playing movies, editing large pictures, and running office applications.

We find the following: (a) the real limitation to NAND Flash memory performance is not its low per-device bandwidth but its internal core interface; (b) NAND Flash memory media transfer rates do not need to scale up to those of HDDs for good performance; (c) SSD organizations that exploit concurrency at both the system and device level (e.g. RAID-like organizations and Micron-style "superblocks") improve performance significantly; and (d) these system- and device-level concurrency mechanisms are, to a significant degree, orthogonal: that is, the performance increase due to one does not come at the expense of the other, as each exploits a different facet of concurrency exhibited within the PC workload.

Categories and Subject Descriptors

B.3 Memory Structures; B.4 I/O and Data Communications;
C.4 Performance of Systems

General Terms

Measurement, Performance, Design, Experimentation

* Cagdas Dirik is a Senior Software Quality Engineer at MicroStrategy. Email: cdirik@microstrategy.com

Keywords

Storage Systems, Flash Memory, Solid State Disks, Performance

1. INTRODUCTION

Flash-based solid state disks are rapidly becoming a popular alternative to hard disk drives as permanent storage, particularly in netbooks, notebooks and PCs, because of flash's faster read access, low power consumption, small size, shock resistance and reliability compared to hard disks. SSDs are commercially available in numerous commodity PC models today; they are considered a high-end option due to a price-per-bit that is higher than HDDs, but that price gap is closing very quickly.

Flash technology has additional characteristics that have slowed its takeover of hard disks, including a bit density that is low relative to HDDs, a limited endurance (i.e., its limited number of write cycles), and its write performance. Solutions have reached a level of maturity to place Flash on a near-term crossover with disk. Rapid migration to later technology nodes and development of multilevel-cell technology have been driving the bit cost of NAND Flash significantly lower and its density higher. NAND Flash capacity has doubled every year since 2001 and is expected to continue at that rate until 2010; by 2010 it is expected to reach 32/64 Gb single chip density [17, 24, 31]. Over the same period, NAND-Flash cost has been decreasing 40-50% per year [29]. In addition, technological enhancements and architectural mechanisms have improved Flash endurance—currently, NAND Flash from several different vendors is commercially available having an endurance rating of more than 50 years at 50 GB write per day. Soon, the limit on the number of writes will become a fading memory (pun intended).

What has received relatively little attention is the interplay between SSD organization and performance, including write performance. As previous studies have shown [8, 1], the relationship between memory-system organization and its performance is both complex and very significant. Very little has been published on the internals of solid-state disk drives; less has been published on the performance resulting from the various design options. The most in-depth study to date has been by Agrawal et al. [1], who analyze different mapping and ganging/striping policies at the device level (i.e., assuming a flash device exported multiple array-select lines to enable concurrent access within the device) and ganging at the system level, targeting both enterprise workloads and synthetic workloads.

By contrast, this study explores in detail the system-level organization choices for solid-state disks—we study a full design space of system-level organizations, varying number of busses, speeds and widths of busses, and degree of concurrent access allowed on each bus. To compare with system-level details, we also investigate device-level design trade-offs as well, including pin bandwidth and I/O width. The design tradeoffs are studied in the context of user-driven workloads. Our study addresses the following issues:

- *Concurrency*. By *system-level organization* we mean the design of the SSD, treating the individual flash devices as constants. Variables in this space include the number of independent busses, their organizations (widths, speeds, etc.), banking strategies, and management heuristics that connect the SSD's flash controller to the flash devices. As shown by Agrawal et al., increasing the level of concurrency in the flash SSD system by striping across the planes within the flash device can amortize the write overhead and increase throughput significantly. Concurrency has been shown in the HDD space to provide tremendous bandwidth increases in interleaved organizations (e.g., RAID); flash is interesting because, unlike disks, its form factor need not change when accommodating interleaved organizations: one can achieve significant levels of concurrency in an SSD without significantly changing its overall size and shape. We investigate the effects of concurrent access to different flash banks via the same channel or by replicating resources and providing multiple independent channels to different flash banks, or by a combination of two.

- *Bandwidth issues*. Common wisdom holds that SSD performance is limited by its media transfer rate. Currently access to a single flash memory chip is provided by an 8-bit bus, which limits the available bandwidth to the 25–50 MB/s range (e.g., 30 ns bus speed, 33 MB/s is common) for read access. For write requests, single chip bandwidth can be much lower at 6-10 MB/s due to slow programming time (200 μs for programming a 2KB page). As interface transfer rates are increasing with the introduction of serial I/O interfaces and fiber channel, HDD performance will continue to scale, but SSD performance is expected to be limited by the device's media transfer rate. Samsung's solution to this problem has been to move to a wider and higher performance bus, which can sustain 108 MB/s (16 bit, 54 MHz). Other vendors have followed suit. Two to three years ago, an 8-bit bus clocked at 50 ns was typical, whereas today most flash solid state disks come with clocks speeds of 20–30 ns. There is also push by other vendors in improving read/write performance of flash disks by access via 800 MB/s bus in a ring topology [15].

- *Write performance*. Another approach in improving flash performance is to reduce the programming time, thus improving the throughput of write requests. For example Micron proposed using two-plane flash devices which can simultaneously read and program two pages (2 KBytes each) in the same flash die [25]. This effectively doubles sustainable read and write bandwidth (reported page program performance increases from 8.87 MB/s to 17.64 MB/s). Another approach taken by Micron is combining flash memory blocks into so-called superblocks, enabling the simultaneous read or write of 2 or 4 pages within a flash device or even across different flash dies [23]; this mechanism is similar to Agrawal's ganging and striping mechanisms. Samsung offers a similar architecture to hide programming latency wherein the flash controller controls 2 separate channels and supports 4-way interleaving (write throughput of 30 MB/s is reported) [28, 29].

The obvious question is *which of these issues is the most significant*—i.e., *what approaches to improving performance provide the best performance at the lowest cost?* In this paper we present a simulation-based performance study of NAND Flash SSD architectures and measure the effectiveness of each of the mentioned mechanisms. We model various flash solid state disk architectures for a typical portable computing environment and quantify their performance under diverse user applications such as browsing, listening to music, watching video, editing pictures, editing text and document creation, office applications and email applications. We find the following:

- The flash memory bus does not need to scale up to HDD I/O speeds for good performance. Average read response times, a good indicator of system-level CPI [18, p. 52], do not improve much beyond 100 MB/s bus bandwidth. The real limitation to flash memory performance is not its bus speed but its core interface—the movement of data between the flash device's internal storage array and internal 2KB *data* and *cache* registers.

- SSD organizations with significant concurrency (e.g. RAID-like organizations) do improve performance significantly. Methods that increase available concurrency by using multiple independent flash banks per channel combined with multiple independent channels provide a scalable storage system. Moreover, there is potential for further improvements by flash-oriented queuing algorithms, access reordering and bus ordering algorithms to accommodate asymmetric read and writes.

Our simulator is based on DiskSim v2.0 [13] and extends its capabilities by implementing NAND Flash memory read/write/erase protocols while still emulating a block-device interface. We model 32 GB NAND Flash SSD system in various organizations by using 1 to 16 flash memory chips connected with 1, 2 or 4 channels with data widths of 8, 16 or 32 bits each, representing bandwidths of 25 to 400 MB/s. We focus on user level workloads in a laptop environment which represent everyday user-level tasks; details of the workload including several trace snapshots are given in Section 4).

2. RELATED WORK

Min and Nam described basics of flash memory and its technological trends [24]. They also outlined various enhancements in the performance of flash memory such as write request interleaving and the need for higher bus bandwidth. Birrell et al. investigated the write performance of flash disks by running micro-benchmarks for USB flash disks under Windows; they noted an increased latency for non-sequential writes [4]. In a similar study Gray and Fitzgerald [14] tested 32 GB Flash SSD from Samsung and reported average request time of 37 msec for 8 KB non-sequential writes. Dumitru [10] provides a comparison of Flash SSDs from various vendors and suggests techniques such as write caching to improve performance. Park et al. proposed high performance controllers for NAND SSDs which can support 2 channels and up to 4-way interleaving in order to hide write latency [29]. Kim and Ahn implement a RAM buffer (similar to write buffers in hard disks) to improve latency of random writes [20]. Additionally, there are many studies that focus specifically on flash memory erase performance and wear leveling. These works look at various ways to hide erase latency, to minimize cost of block cleaning and to ensure uniform wear leveling across flash memory blocks [2, 7, 32].

Flash solid state disks emulate a block-device interface. A different approach is using a file system specific for flash memory and letting system software manage flash storage [9, 12, 19, 22]. These file systems usually employ a log-structured approach [30]. A survey on flash specific file systems and related patents can be found in [11]. This survey also discusses various sophisticated data structures and algorithms designed to overcome the limitations of flash memory.

Another use for flash memory in storage systems is in a hybrid setting. Bisson and Brandt use flash memory as a non-volatile cache called NVCache to provide an extension to hard disks [5]. By using flash memory with hard disk drives in a hybrid configuration, storage sub-systems I/O performance is improved and power consumption is reduced. This is consistent with similar results from Jacob, Ng, and Wang [18]. Similar studies provide variations of hybrid storage systems which use both flash memory and hard disk [6, 21].

Recently, Myers discusses use of NAND Flash memory in relational databases. He investigates using log-structured file systems for improved write performance and focuses on available parallelism in multi-chip flash solid state disks. His work concludes that current flash memory technology is not mature enough to be used in high-performance relational databases [26].

Finally, Agrawal et al. provide a detailed discussion on design tradeoffs for NAND Flash SSDs [1]. Their work analyzes different SSD organizations using synthetic workloads and enterprise traces. In their work, the serial interface to flash memory is considered the primary bottleneck for performance. By employing parallelism within a flash memory package and interleaving requests to a flash memory die, the overall system bandwidth is doubled. Although our study resembles theirs, there are major differences on the methodology and area of investigation between their work and ours. One of their conclusions is that SSD performance is highly workload sensitive: they find that performance differs substantially if write requests are sequential or random (their synthetic traces are largely sequential; their enterprise traces are largely random; the performance improvements shown for the synthetic traces are far more significant than those shown for the real-world traces). Additionally, the workloads used in their study are read oriented, with roughly a 2:1 read-to-write ratio, which helps to hide the problem of slow writes in an SSD. However, in PC applications (user-driven workloads), there tends to be a much higher proportion of writes: in our workloads, we see a 50:50 ratio, which would tend to expose flash's write problem. User driven workloads are not biased towards sequential or random requests but provide a mix of random and sequential writes at a given time interval (see Figure 4). Agrawal's study outlines core limitations of flash memory within the boundaries of a flash memory device/package—limitations such as logical to physical mapping granularity, limited serial interface, block erasure, cleaning frequency and wear leveling. Our study extends their work by focusing on exploiting concurrency in SSD organizations at both the system and device level (e.g. RAID-like organizations and Micron-style superblocks). These system- and device-level concurrency mechanisms are, to a significant degree, orthogonal: that is, the performance increase due to one does not come at the expense of the other, as each exploits a different facet of concurrency available within SSD organizations.

3. BACKGROUND

NAND Flash memory is a type of nonvolatile electrically erasable programmable read only memory (EEPROM) where memory cells are connected in series between ground and bit lines. This cell organization allows NAND Flash memory to have much smaller cell area and bit cost and to consume less power [3, 27]. Data can be read from and written into NAND Flash memory using an indirect I/O like interface via an 8-bit bus, which is used for both data and address information, and for issuing commands. Though the 8-bit I/O bus can be considered a limitation, it allowed industry-wide standardization of pin counts for NAND Flash memory packages, and this commoditization helped propel the technology into its current market position. In addition to read and write operations, NAND Flash memory also supports a third type of operation, the erase command. Flash memory technology does not allow overwrite of data (in-place update of data is not allowed) since a write operation can only change bits from 1 to 0. Therefore, to change a cells value from 0 to 1, one has to erase a group of cells first by setting all of them to 1.

3.1. NAND Flash Array Structure

NAND Flash memory is organized into blocks where each block consists of a fixed number of pages. Each page stores data and corresponding metadata and ECC information. A single page is the smallest read and write unit. Earlier versions of flash memory had page sizes of 512 Bytes and block size of 16 KBytes (32 pages). Currently a typical page size is 2 KBytes (4 sectors of 512 Bytes each), and a typical block size is 128 KBytes (64 pages). The number of blocks and pages vary with the size of the flash memory chip. In addition to storage cells for data and metadata information, each flash memory die includes a command register, an address register, a data register and a cache register. Figure 1 shows NAND Flash memory array organization for a sample 1 Gb flash memory from Micron [25].

3.2. Read/Write Command

In NAND Flash memory, the smallest access unit is a page. To read a page one issues a read command to the command register

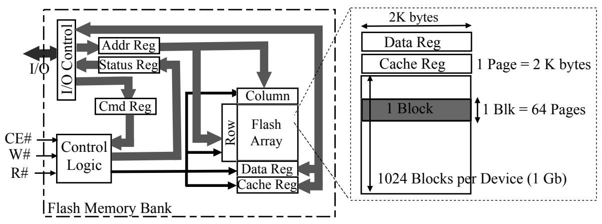
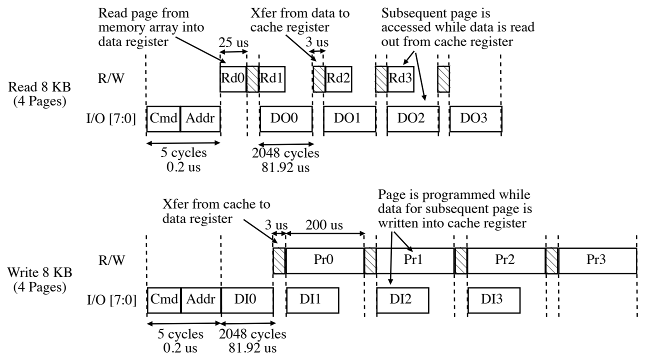


Figure 1: NAND Flash Memory Array Organization. A typical 1 Gb flash memory array consists of 1024 blocks. Each block contains 64 pages of 2 KB each. Figure adapted from [25].



Read page from memory array into data register
Xfer from data to cache register
Subsequent page is accessed while data is read out from cache register
25 us
3 us
Read 8 KB (4 Pages)
R/W
Rd0
Rd1
Rd2
Rd3
I/O [7:0]
Cmd
Addr
DO0
DO1
DO2
DO3
5 cycles 0.2 us
2048 cycles 81.92 us
Xfer from cache to data register
Page is programmed while data for subsequent page is written into cache register
3 us
200 us
Write 8 KB (4 Pages)
R/W
Pr0
Pr1
Pr2
Pr3
I/O [7:0]
Cmd
Addr
DI0
DI1
DI2
DI3
5 cycles 0.2 us
2048 cycles 81.92 us

Figure 2: Flash memory Read and Write Operation Timing Diagram. Read/write 8 KB with 25 µs page read time, 200 µs page program time and 25 MHz 8-bit I/O bus. Page size is 2 KBytes. Cache mode operation is typical when accessing sequential pages.

and then writes a block number and page number within the block into the address register. The complete page data (2 KBytes) will be accessed in 25 µs and will be loaded into the data register. Afterwards, data can be read from the data register via the 8-bit I/O bus. If sequential pages need to be accessed within a block, the read command can be used in cache mode. In this mode, when the first page is loaded into data register, it will be transferred from data register to the cache register. Typically, copying data from data register to cache register takes 3 µs. While data is read out from cache register via 8 bit I/O bus, a subsequent page can be read into the data register.

Similar to a read command, a write or program command must be issued at the page level, and pages within a block are written in sequential order. To program a page, one issues a write command to the command register, writes a block number and page number into the address register and loads data into the data register. The data is programmed into the page in 200 µs. To program more than one page, the write command can be used in cache mode , which is similar to the read-command cache mode described earlier and allows concurrent access via the cache and data registers.

Figure 2 shows the timing of sample read and write commands for 8 KByte read and write requests. The timing of read requests is heavily dependent on I/O bus speed, while the timing of write requests is determined by how quickly a page can be programmed.

1. Erase Command

As mentioned, a write operation in flash memory can only change bit values from 1 to 0. The only way to change bit values from 0 to 1 is by erasing. Unlike read and write commands, the erase command can only be performed at the block level. Once issued, all bit values in all pages within a block are set to 1. To erase a block, one issues the erase command to the command register and loads a block number into the address register. Then flash memory will set its status to busy for 2 ms while the erase operation is performed and verified.

2. LBN-PBN Mapping and Block Cleaning

NAND Flash memory does not allow in-place update of data. Once a page is written, subsequent writes to that page cannot proceed until the block in which the page resides is erased. Since it is clearly too expensive to perform an erase for every write, flash memory allocates to write requests a different, newly erased page and redirects the write request (and any subsequent read requests) to this new page. Thus, flash devices must manage a logical to physical address mapping.

Over time as more write requests are serviced, the number of pages with invalid data increases, and the number of newly erased pages decreases. To service more write requests, blocks with invalid pages need to be cleaned via the erase operation. This is a garbage-collection process and is managed by the flash controller, external to the devices.

3. FTL and Host Interface

Compared to HDD, flash memory provides a simpler read/write interface, one without the complexities of mechanical parts. On the other hand, flash memory has its own peculiarities of block erasing and logical-physical address mapping. To use flash memory as a storage device, one needs to hide these peculiarities from the host system, since file systems and virtual memory systems assume a block device interface when accessing storage. For this purpose, Flash SSDs implement a software layer called Flash Translation Layer that emulates an HDD for host systems. Figure 3 shows a sample 32 GB NAND Flash SSD architecture from Samsung [28]. When the host system calls for a block read or write, the host interface layer in Flash SSD interprets the host's command and invokes associated FTL layer functions. The FTL converts the logical block address into a physical page address in the flash memory and initiates read, write or erase commands in NAND interface layer. In addition to logical to physical address mapping, FTL is also responsible for implementing all features of flash memory such as block management, erase unit reclamation, wear leveling and internal data movements. A survey of algorithms and data structures implemented at FTL layer can be found at [11]. Finally, the NAND interface layer implements internal flash commands and accesses data in flash memory array.

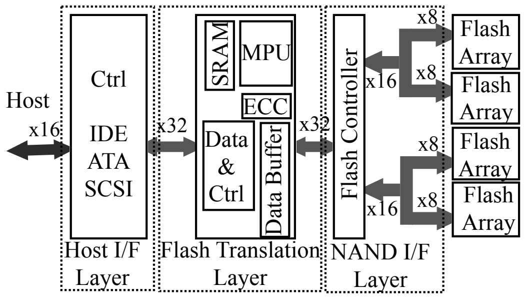


Figure 3: Organization of a conventional 32 GB NAND Flash SSD. Figure adapted from [28].

2. EXPERIMENTAL METHODOLOGY

For accurate timing of disk requests for a flash SSD, we integrated our flash memory code into DiskSim v2.0. DiskSim is an efficient, accurate disk system simulator from Carnegie Mellon University and has been extensively used in various research projects studying storage subsystem architectures [13]. Our modified version of DiskSim can simulate a generalized Flash NAND SSD by implementing flash specific read, program, erase commands, block cleaning, LBN-PBN mapping, all while providing the illusion of representing an HDD.

In this study, we have modeled a 32 GB ATA 133 MB/s NAND Flash SSD with the following timing parameters: page access time of 25 µs, page program time of 200 µs and block erase time of 3 ms. We have assumed page sizes of 2 KBytes and block sizes of 64 pages. Logical to physical address mapping is performed at the granularity of a page. To measure the impact of media transfer rate, we varied the speed at which data can be read from the flash devices to the external controller (over the flash device's external pins). We modeled 8-, 16- and 32-bit wide I/O busses at speeds of 25, 50 and 100 MHz.

We wanted to model today's typical SSDs which usually support 2 channels and up to 4-way interleaving (e.g., the configuration modeled in [1]). Therefore we simulated various configurations of flash memory banks on a shared bus or multiple independent channels to different flash banks or a combination of the two. In our simulations each flash memory bank (either a single device or a set of interleaved devices, similar to Agrawal's ganging) can accept one request at a time and can operate independently of other banks unless they are linked together in Micron-style superblocks. The size of each flash memory bank can change, as the entire storage capacity is kept constant at 32 GB. For example, if 4 banks are connected via a single shared bus, then each bank is 8 GB in size. If a configuration with 4 independent I/O channels and 4 memory banks per channel is used, then each bank is 2 GB in size (system capacity is 16 x 2 GB).

To simulate a realistic flash management model, we have assumed modular striping for write requests. If we have a total of *x* banks, the Nth write request is assigned to bank number N(mod *x*). We have maintained a pool of free blocks for each bank and have allocated pages from the current working free block when a write request is received. For example; if a write request of 8 KBytes is received, data is written into the next available 4 pages within the current working free block and logical-to-physical address mapping is updated accordingly.

2.1. I/O *Workloads*

DiskSim can be used as a trace-driven simulator or can internally generate synthetic workloads [13]. In our study, we have used our own disk traces collected from portable computers and PCs running real user workloads. Our workloads represent typical multi-tasking user activity, which includes browsing files and folders, emailing, text editing and document creation, surfing the web, listening to music and playing movies, editing pictures, and running office applications.

The characteristics of our traces are consistent with expected I/O traffic for personal computer workloads reported by Hsu and Smith [16]. The average I/O per second in our traces ranges from 1.6 Mbps to 3.5 Mbps, which is similar to 2.37 Mbps reported in [16]. Our personal computer workloads generate 4.6 to 21.35 I/O requests per second with an average request size of 26 KB. Although this average request size is much higher than 7-9 KB expected by [16], it is weighted by a small number of large files: approximately half of the requests generated in our traces are 4-8 KB. We observed that average request size in our personal workloads is skewed by occasional very large write requests (of size 64 KB and higher). We have also confirmed that I/O traffic in our workloads is bursty, localized and balanced—I/O requests arrive in groups, frequently access localized areas of the disk, and are partitioned roughly 50:50 between reads and writes. Figure 4 summarizes properties of our traces and shows three different 8-minute snapshots from three different traces, representing different mixes of reads and writes.

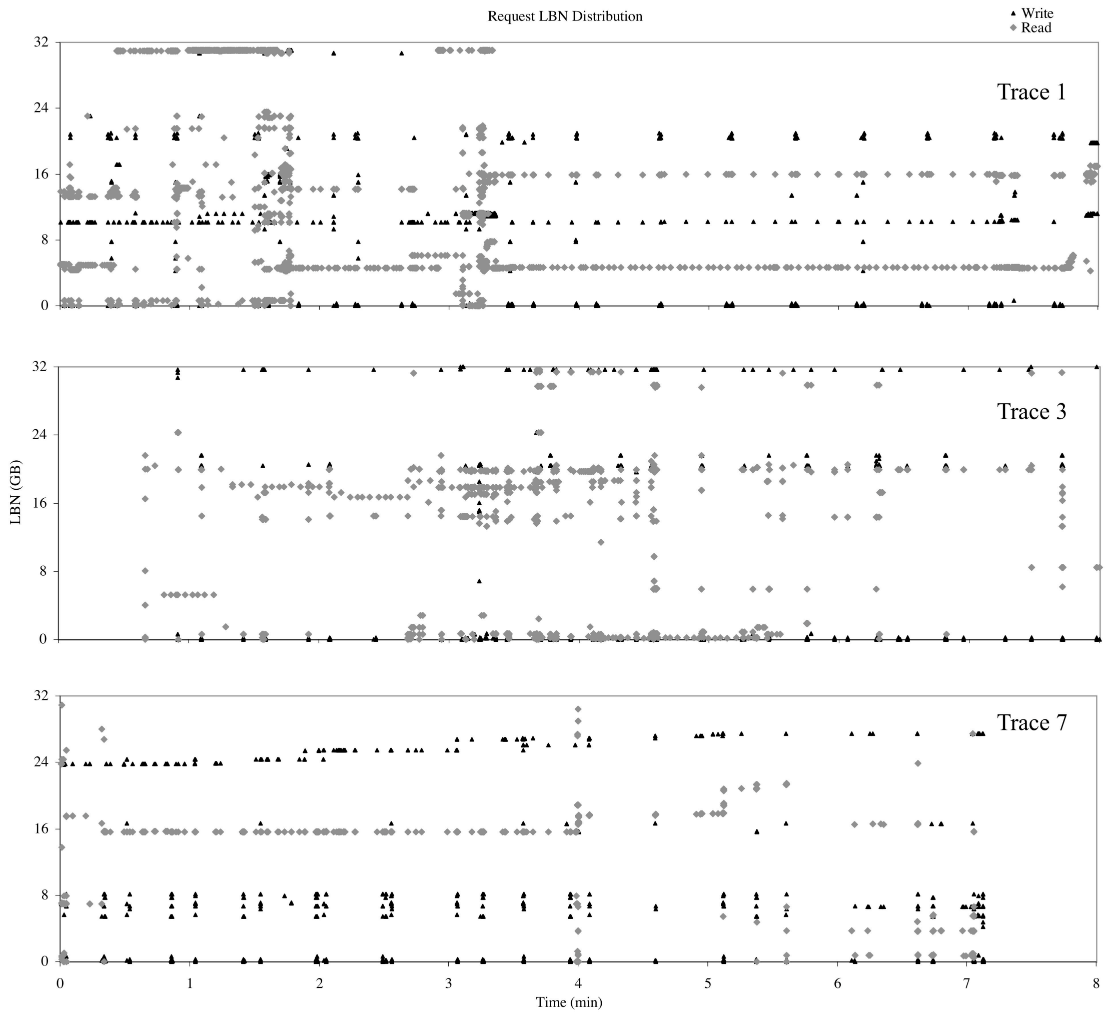
2.2. Average Read Latency

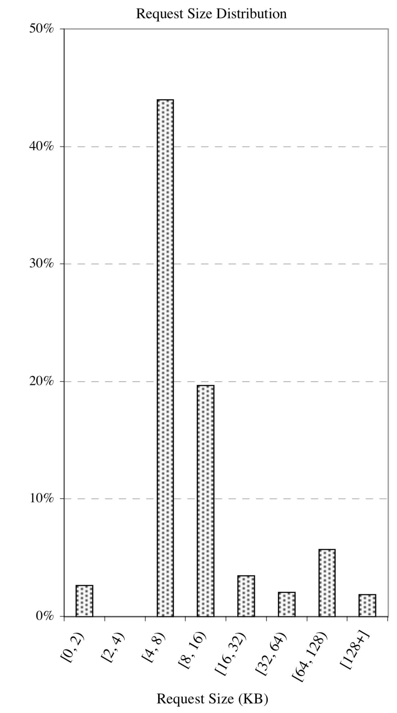
We present performance in terms of average *request* response time, average *write* request time, and average *read* request time. As shown in Jacob, Ng, & Wang [18 p. 52], overall computer-system performance (i.e., CPI) tracks disk's average *read* response time and not the disk's average *request* response time, which includes both reads and writes. The observation is true for both read-dominated applications and applications with significant write activity. In our simulations, write-request behavior is modeled in detail to address flash's write-performance issue as well as to determine its effect on read performance.

3. EXPERIMENTAL RESULTS

3.1. Banking and Interleaving

One way to hide write (program) latency in flash memory has been interleaving sequential writes by dividing the flash array into banks where each bank can read/write/erase independently. Since flash memory allocates a newly erased page for a write request, choosing an empty page for each write becomes a run-time decision of resource allocation. When sequential write requests arrive, one can assign pages for these writes from different banks. This way sequential writes can be dispatched to multiple independent banks in parallel, and page write times can be





	Trace Duration (minutes)	Avg. Number of I/O per sec	Avg. Number of Mbits per	Read Percentage	Write Percentage
Trace 1	145	11.76	1.74	59.93%	40.07%
Trace 2	62	11.29	3.00	52.92%	47.08%
Trace 3	75	21.35	3.39	43.44%	56.56%
Trace 4	113	7.50	1.60	52.32%	47.68%
Trace 5	143	10.06	1.82	44.52%	55.48%
Trace 6	155	4.58	1.39	52.63%	47.37%
Trace 7	44	8.30	3.48	33.99%	66.01%
Overall	737	9.99	2.04	49.81%	50.19%

Figure 4: I/O Workloads. Properties of our user driven disk traces are listed. Snapshots from different traces confirm bursty, localized and read/write balanced I/O traffic.

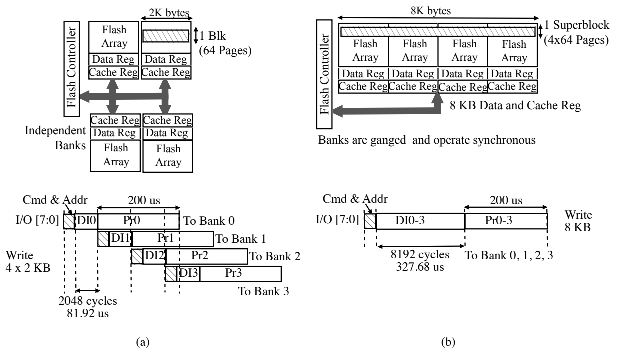


Figure 5: Banking and Superblocks. (a) 4-way baking and timing diagram of 4 subsequent write requests of 2 KB each. I/O bus is shared by independent banks. (b) 4-way superblocks and timing diagram of writing 8 KB. Blocks across blocks are linked together to create superblocks. Data and cache registers are linked as well.

interleaved. Figure 5(a) shows a flash array organization with 4-way interleaving and timing diagram for 4 sequential write requests of 2KB each.

Figure 6(a) shows the effect of increasing the degree of banking on average disk-request response time. Note that response time is sum of physical access time (time to read/write data to/from flash array) and queue wait time. One sees significant improvements in both read and write request times, 50–80%, when the level of banking is increased from 1 to 2 and 2 to 4. However, from 4- to 8-way banking, reads and writes start to show different performance characteristics. While request times continue to improve for writes, read-request performance starts to flatten, moving from 4 to 8-way banking. This is explained by an increase in the physical access times at high levels of banking due to bus contention—especially for low bandwidth 8-bit 25 MHz bus configurations. The more banks per channel, the larger the degree of bus utilization, to the point of traffic congestion. As shown before in Figure 2, read request timing mostly consists of time spent in reading data from the I/O bus and is thus more sensitive to degradation in the I/O channel than writes; any congestion in the I/O bus will impact reads more than writes. Performance of read requests is critical since overall system performance tracks disk's average read response time [18]. Therefore one does not gain much by increasing interleaving from 4 to 8 in a single channel configuration. If 8-way banking is supported by 2 channels rather than a single channel, read performance does improve 20%. 4-way banking is the optimum level of concurrency on a single I/O bus for these workloads.

3.2. Superblocks

Another way to hide write latency in flash memory and to improve both read and write performance is to gang blocks across individual flash banks to create superblocks [23]. Individual flash memory banks are combined by sharing chip-enable, command signals, and I/O signals. Sharing command signals enables merging physical blocks across flash arrays to create a designated superblock. This effectively increases the size of available data and cache registers and enables the superblock to process a higher volume of data in one step. Figure 5(b) shows a sample flash array organization with 4-way superblocks.

The effect of superblocks on average response time is shown in Figure 6(b). Similar to banking, superblocks also improve performance significantly in 2- and 4-way configurations, with diminishing returns in read performance for 8-way configurations. Compared to banking, superblocks provide 10–60% better performance, especially at higher I/O bandwidths. When the I/O bandwidth is low and the level of interleaving is high, banking outperforms superblocks by 27% in servicing write requests, as shown in Figure 6(c).

Improving performance by banking is relatively cheap—independent banks on a shared bus do not increase controller complexity, and bus arbitration can be implemented with low cost. On the other hand, superblocks require additional controller complexity in managing blocks across flash memory banks, as each block must be checked independently (for example, compare Figures 5 & 6 in [23]). Another trade-off with superblocks is the fact that blocks are linked together permanently, due to the hardwiring of control and I/O pins from multiple devices. If any one of the blocks in a superblock becomes bad, all blocks in that superblock are considered unusable, thus reducing available storage. This represents a significant design trade-off: superblocks provide better performance than banking, but banking is a more reliable solution and is thus potentially more scalable.

3.3. Media Transfer Rate

One of the factors limiting flash memory performance is believed to be its media transfer rate. In current flash devices, 8-bit 33MB/s I/O buses are common. As HDDs with 7200 or 10K RPM are popular, and disk-interface speeds are scaling up with serial interface and fiber channel, SSD performance is expected to be limited by the media transfer rate. We have measured the effect of media transfer rate on the performance of NAND Flash SSD by scaling I/O bus bandwidth from 25 MB/s (8-bit wide bus at 25 MHz) up to 400 MB/s (32-bit wide bus at 100 MHz). As shown in Figure 7, performance does not improve significantly beyond 100 MB/s.

However, note that, even when performance saturates at high bandwidths, it is still possible to achieve significant performance gains by increasing the level of concurrency by either banking or implementing superblocks. Performance saturates at 100MB/s because the real limitation to NAND Flash memory performance is the device's core interface—the requirement to read and write the flash storage array through what is effectively a single port (the read/cache registers)—and this is a limitation that concurrency overcomes.

3.4. Increasing the Degree of Concurrency

As shown previously, flash memory performance can be improved significantly if request latency is reduced by dividing the flash array into independent banks and exploiting concurrency. The flash controller can support these concurrent requests through multiple flash memory banks via the same channel or through

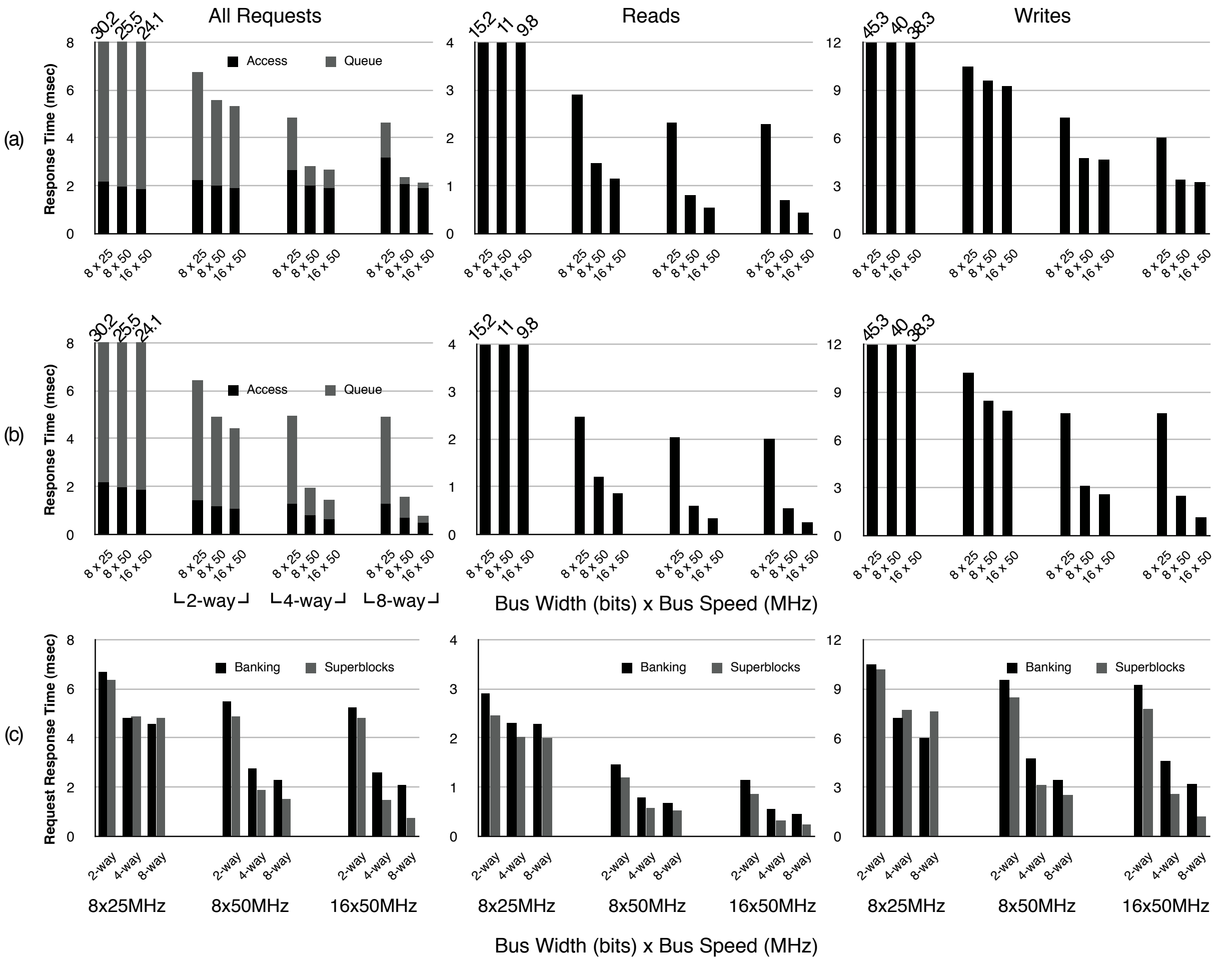


Figure 6: Response Times: Banking vs. Superblocks. (a) 2, 4, and 8-way banking; performance dramatically increase when the level of banking is increased from 1 to 2 and 2 to 4. Read-request performance starts to flatten, moving from 4 to 8-way banking. (b) 2, 4, and 8-way superblocks. Performance improves significantly in 2- and 4-way configurations, with diminishing returns in read performance for 8-way configurations. (c) Comparing banking against superblocks with varying I/O bus width and speed.

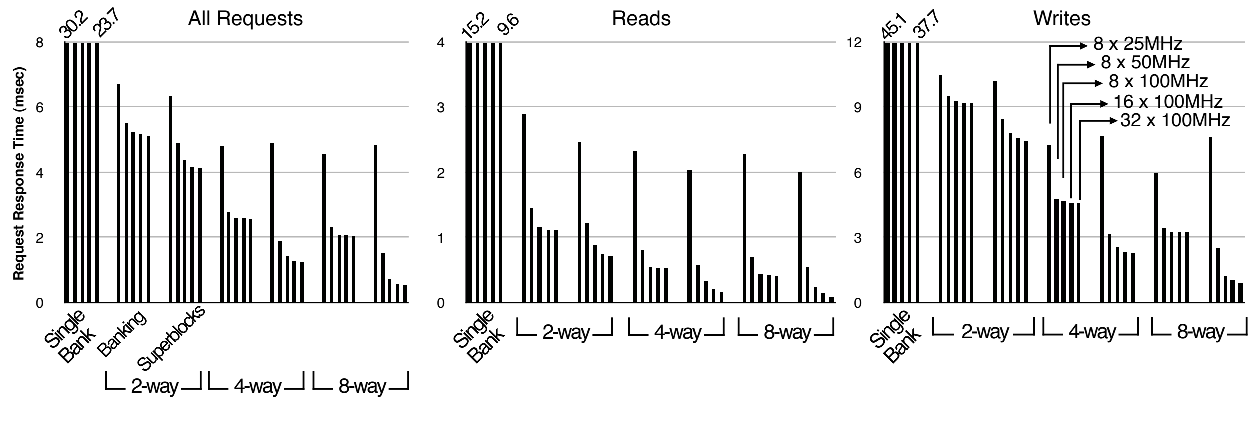
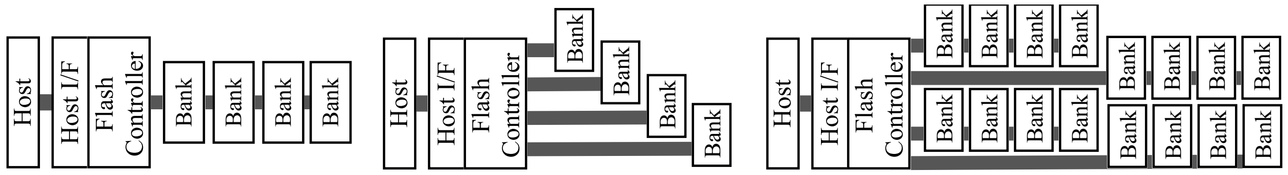


Figure 7: Media Xfer Rate - Changing I/O bandwidth from 25 MB/s (8bit x 25 MHz bus) to 400 MB/s (32 bit x 100 MHz bus). Performance does not improve significantly beyond 100 MB/s.



(a) Single channel (b) Dedicated channel for each bank (c) Multiple shared channels

Figure 8: Flash SSD Organizations. (a) Single I/O bus is shared - 1, 2, or 4 banks; (b) dedicated I/O bus: 1, 2, or 4 buses and single bank per bus; (c) multiple shared I/O channels - 2 or 4 channels with 2 or 4 banks per channel.

multiple independent channels to different banks, or through a combination of the two. To get a better idea of the shape of the design space, we have focused on changing the degree of concurrency one I/O bandwidth at a time. Figure 8 shows example configurations modeled in our simulations with bandwidths ranging from 25 MB/s to 400 MB/s. This is equivalent to saying, "I have four 50 MHz 8-bit I/O channels ... what should I do? Gang them together, use them as independent channels, or try some combination of the two?"

The performance results are shown in Figure 9. Though increasing the available concurrency in the storage sub-system (number of banks x number of channels) typically increases performance, it does not always do so by very much. For example, consider Figure 9(a); comparing 4 banks with 2 channels against 4 banks with 4 channels, adding 2 channels does not improve performance if the channels are already fast enough.

Trying to exploit concurrency by splitting an I/O bus into multiple narrow channels does not improve performance. Moreover, if the I/O bus is slow, it has a negative effect, as is shown in Figure 9(b). Read and write requests show different trends. When total I/O bandwidth of the system is fixed, as in Figure 9(c), read requests prefer faster channels. On the other hand write requests prefer multiple channels. Given a storage sub-system with fixed media transfer bandwidth, there are always several near-optimal configurations that are within several percent of each other.

1. Queueing and Read Priority

As mentioned before, read and write requests in flash memory show different characteristics. One major difference is their asymmetric nature, as shown in Figure 2. Read request performance is heavily dependent on I/O bus width and clock speed. On the other hand, write requests are limited by core programming time. Another difference is the scale factor in their timing: a write request usually executes 2–8 times slower than a read request, depending on the I/O bandwidth. These differences provide an additional opportunity to improve storage sub-system performance by giving priority to reads in request scheduling. For example, assume read access takes x and a write access takes $4x$ amount of time, and a read request is received immediately following a write request. Simply scheduling these requests in the order they are received will result in average response time of 4.5x. However, if the read request is given priority and issued earlier, the average response time will be 3x. Moreover, read performance will improve by a factor of 5 while write performance is only effected by 25%. When combined with the fact that overall system performance tracks disk's average read response time (not the disk's average response time) for both read-dominated applications and applications with significant write traffic, one can simply improve NAND Flash SSD performance by giving priority to read requests over write requests.

As shown in Figure 10, we have simulated effects of read priority on performance of flash memory with superblocks. When reads are given priority over writes, their performance improves significantly, by 30–50%. At the same time, writes show a slight performance degradation of roughly 5%. Superblocks provide a good example because of their superior write performance—read and write timings show the smallest scale of difference. For other memory organizations such as banking and multiple channels, the impact will much bigger as asymmetry between reads and writes increase.

2. CONCLUSIONS

We have simulated various NAND Flash SSD architectures for a typical portable computing environment and found that NAND Flash memory performance is not limited by its serial interface. Given a storage sub-system with fixed media transfer bandwidth, there are several configurations that result in excellent performance without significant costs—i.e., one need not move into GB/s bandwidths or 128-bit data widths to achieve good performance. SSD organizations that exploit concurrency at both the system and device level (e.g. RAID-like organizations and Micron-style superblocks) improve performance significantly. Due to the asymmetric nature of read and write requests, one configuration that provides the best write performance may not be the best choice for read performance. Moreover, asymmetry between reads and writes provide potential for further performance improvements by flash oriented queueing algorithms.

REFERENCES

[1] Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J. D., M. Manasse, and Panigraphy, R. 2008. Design Tradeoffs for SSD Performance. In *Proceedings of the USENIX Annual Technical Conference* (Boston, MA, June 2008). USENIX 2008.

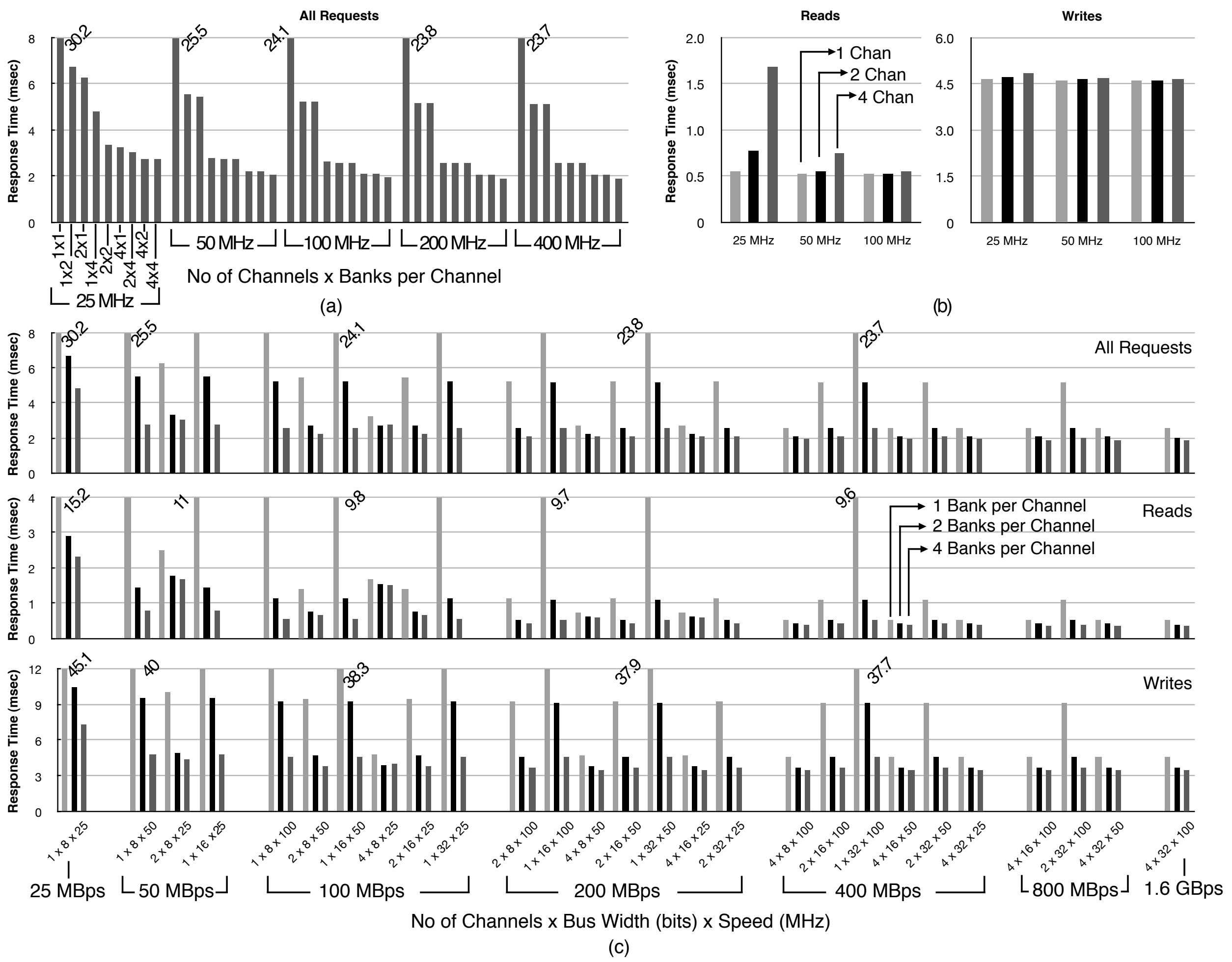


Figure 9: Level of concurrency. (a) Increasing concurrency by using multiple 8-bit channels and multiple banks per channel; (b) splitting single 32-bit 25 MHz I/O bus into 2 16-bit channels or 4 8-bit channels; (c) changing bank and channel organization for fixed I/O bandwidth.

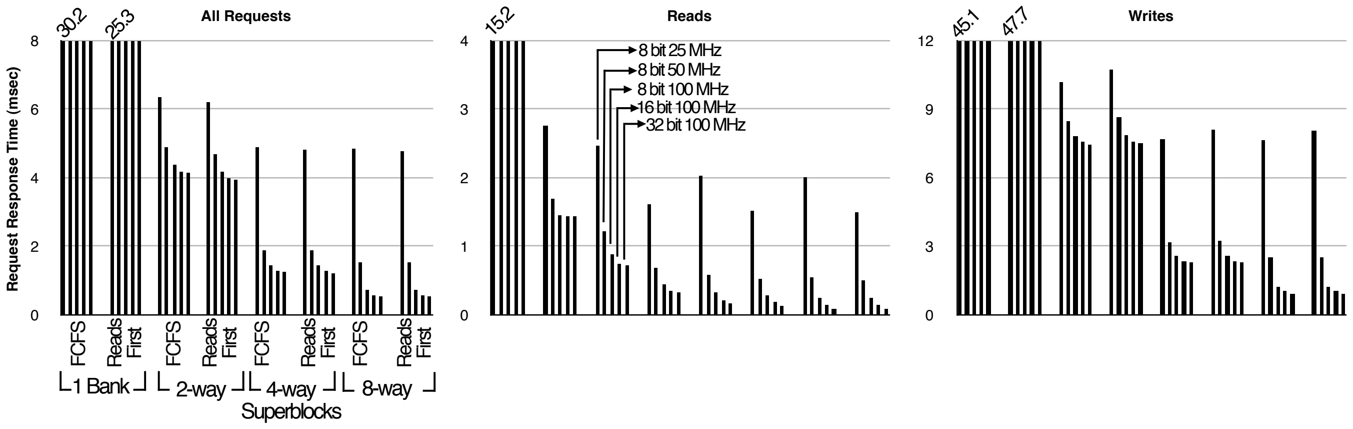


Figure 10: FCFS vs. Read Priority. Comparing first come first serve scheduling policy against reads first in 2, 4, and 8-way superblocks.

[2] Baek, S., Ahn, S., Choi, J., Lee, D., and Noh., S. H. 2007. Uniformity Improving Page Allocation for Flash Memory File Systems. In *Proceedings of the 7th ACM & IEEE International Conference On Embedded Software* (2007), 154-163.

[3] Bez R., and Cappelletti P. 2005. Flash Memory and Beyond. In *2005 International Symposium on VLSI Technology* (April 2005). IEEE VLSI-TSA, 84-87.

[4] Birrell, A., Isard, M., Thacker, C., and Wobber, T. 2007. A Design for High-Performance Flash Disks. *ACM SIGOPS Operating Systems Review*, vol. 41, no. 2, 88-93.

[5] Bisson, T., and Brandt, S. A. 2007. Reducing Hybrid Disk Write Latency with Flash-Backed I/O Requests. In *Proceedings of the 15th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. MASCOTS'07.

[6] Chen, F., Jiang, S. and Zhang, X. 2006. SmartSaver: Turning Flash Drive into a Disk Energy Saver for Mobile Computers. In *Proceedings of the 11th International Symposium on Low Power Electronics and Design, (*October, 2006). ISLPED'06.

[7] Chiang, M.-L., and Chang, R.-C. 1999. Cleaning Policies in Mobile Computers Using Flash Memory. *Journal of Systems and Software*, vol. 48, no. 3, 213-231.

[8] Cuppu, V., and Jacob, B. 2001. Concurrency, Latency, or System Overhead: Which Has the Largest Impact on Uniprocessor DRAM-System Performance? In *Proceedings of the 28th Annual ACM/IEEE International Symposium on Computer Architecture* (Göteborg, Sweden, June 2001). ISCA 2001, 62–71.

[9] Dai, H., Neufeld, M., and Han, R. 2004. Elf: An Efficient Log Structured Flash File System for Micro Sensor Nodes. In *Proceedings of the 2nd International Conference on Emdedded Networked Sensor Systems*. SenSys'04, 176-187.

[10] Dumitru, D. 2007. Understanding Flash SSD Performance. http://managedflash.com/news/papers/easyco-flashperformance-art.pdf (August 2007).

[11] Gal, E., and Toledo, S. 2005. Algorithms and Data Structures for Flash Memories. *ACM Computing Surveys*, vol. 37, no. 2, 138-163.

[12] Gal, E., and Toledo, S. 2005. A Transactional Flash File System for Microcontrollers. In *Proceedings of the USENIX Annual Technical Conference*, 89-104.

[13] Ganger, G., R., Worthington, B. L., and Patt, Y. N. The DiskSim Simulation Environment Version 2.0 Reference Manual. http://www.pdl.cmu.edu/DiskSim/disksim2.0.html.

[14] Gray, J., and Fitzgerald, B. 2007. Flash Disk Opportunity for Server-Applications. http://research.microsoft.com/~gray/papers/FlashDiskPublic.doc (January 2007).

[15] HLNAND. HyperLink NAND Flash. MOSAID Technologies Inc., http://hlnand.com/852572C9004980E9/ID/Next-Gen-Memory-WP1, May 2007.

[16] Hsu, W., and Smith, A. J. 2003. Characteristics of I/O Traffic in Personal Computer and Server Workloads. *IBM Systems Journal*, vol. 2, no. 2 (April 2003), 347-372.

[17] Hwang, C. 2003. Nanotechnology Enables a New Memory Growth Model. *Proceedings of the IEEE*, vol. 91, no. 11 (November 2003), 1765-1771.

[18] Jacob, B., Ng, S., and Wang, D. 2007. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann.

[19] JFFS2: The Journalling Flash File System. Red Hat Corporation. http://sources.redhat.com/jffs2/jffs2.pdf, 2001.

[20] Kim, H., and Ahn, S. 2008. A Buffer Management Scheme for Improving Random Writes in Flash Storage. In *Proceedings of the 6th USENIX Symposium on File and Storage Technologies*. FAST'08, 239-252.

[21] Kim, Y., Lee, S., Zhang, K., and Kim, J. 2007. I/O Performance Optimization Techniques for Hybrid Hard Disk-Based Mobile Consumer Devices. *IEEE Transactions on Consumer Electronics*, vol. 53, no. 4 (November 2007), 1469-1476.

[22] Manning, C. 2004. YAFFS: Yet Another Flash File System. http://aleph1.co.uk/yaffs.

[23] Memory Management in NAND Flash Arrays. Micron, Inc. Technical Note TN-29-28. http://download.micron.com/pdf/technotes/nand/tn2928.pdf, 2005.

[24] Min, S. L., and Nam, E. H. 2006. Current Trends in Flash Memory Technology. In *Proceedings of the 2006 Asia South Pacific Design Automation* (January 2006). ASP-DAC '06, 332-333.

[25] MT29F1GxxABB 1 Gb NAND Flash Memory. Micron Technology, Inc., http://download.micron.com/pdf/datasheets/flash/nand/1gb_nand_m48a.pdf, 2006.

[26] Myers, D. 2007. On the Use of NAND Flash Memory in High-Performance Relational Databases. Master's thesis. MIT.

[27] NAND Flash Applications Design Guide. Toshiba America Electronic Components, Inc. http://www.dataio.com/pdf/NAND/Toshiba/NandDesignGuide.pdf.pdf, April 2003.

[28] NAND Flash-based Solid State Disk Module Type Product Data Sheet. Samsung Electronics Co., Ltd., http://www.bigboytech.com/new/v1.5/ssd/docs/ssd_module_type_spec_rev121.pdf, January 2007.

[29] Park, C., Talawar, P., Won, D., Jung, M., Im, J., Kim, S., and Choi, Y. 2006. A High Performance Controller for NAND Flash-based Solid State Disk (NSSD). In *Proceedings of the 21st IEEE Non-Volatile Semiconductor Memory Workshop*. NVSMW, 17-20.

[30] Rosenblum, M., and Ousterhout, J. 1992. The Design and Implementation of a Log-Structured File System. *ACM Transactions on Computer Systems*, vol. 10, no. 1, 26-52.

[31] Shin, Y. 2005. Non-volatile Memory Technologies for Beyond 2010. In *2005 Symposium on VLSI Circuits* (June 2005), 156-159.

[32] Wu, M., and Zwaenepoel, W. 1994. eNVy: A Non-Volatile, Main Memory Storage System. In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS, 86-97.